Team 08 of Section 005
William Morgan (wtmorga3) Jake Pope (jmpope3) Joey Woodring (jjwoodri) Pierce Willoughby (pxwillou)
Connect Four

CSC116: Comprehensive Exercise Report

NOTE: You will replace all placeholders that are given with <<>>

Team 08 of Section 005
William Morgan (wtmorga3) Jake Pope (jmpope3) Joey Woodring (jjwoodri) Pierce Willoughby (pxwillou)
Connect Four

Requirements/Analysis

Instructions: http://go.ncsu.edu/csc116-ce-reqt

Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

1. After reading the client's brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.
    - Software should be a playable game of Connect Four, a game in which players take turns dropping chips into a grid until one of them has four in succession.
        - The game must be won when there are four chips of the same color in succession
        - Chips in succession include horizontal, vertical, and diagonal
        - Each player takes turns placing a chip down any of the vertical columns of the game board
        - The chips must be two distinct characters
        - Players must be updated with a number of chips placed and current max chips in row
        - Grid must be 8 X 8

2. After reading the client's brief (possibly incomplete description), what questions do you have for the client? Are there any pieces that are unclear? After you have a list of questions, raise your hand and ask the client (teaching staff) the questions; make sure to document their answers.
    - How would we make the columns clickable?
    - Will it be a two-player game or single-player against AI?

3. Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
    - Checking items against each other in a 2D array, specifically checking diagonal indexes

4. Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
    - Small children mostly, but anyone can play connect four

5. Describe how each user would interact with the software. What information does the user "input" into the software?
    - Users would click on the column they desire their chip to go in

6. What features must the software have? What should the users be able to do?
    - The user must be able to click on a column with an empty space and place their chip
    - The software must be able to check for a four in a row winning grid
    - Either two-player or a random AI to play against

7. What is the "output" of the software? Is output to the terminal, to a file, or to some other location?
    - Display interactable connect four grid, possibly in GUI form
    - Display a winner screen when someone achieves 4 in succession

8. Other notes:

Software Requirements

Use your notes from above to complete this section of the formal documentation by writing a detailed description of the project, including a paragraph overview of the project followed by a list of requirements (see lecture for the format of requirements). You may also choose to include user stories.

Overview

The connect four game we will be making must be a user interactive game that allows the player to place chips on a playing board until either player has four chips in succession (horizontally, vertically, or diagonally) or the entire board is full with no chips in 4 places. This will be done using two object classes, one for the chip, and one for the playing board. The playing board class will use the chip objects in order to control the logic of the connect four game. Once a player has four chips in a row, a game-winning screen will be displayed and the player has the option to quit, or begin a new game. If the board fills up with no winning patterns, the game will end and the quit and new game options will be available.

List of Requirements

- The game must be won when there are four chips of the same color in succession
- Chips in succession include horizontally, vertically, or diagonally
- Each player takes turns placing chips down the vertical columns
- Once one chip has been placed, another cannot be added in its place
- Chips must be two distinct characters
- If a column is full, that column should no longer be interactable
- Chips must be placed at the bottom first then stack on top of each other
- Game updates players as the game is played
- Grid must be an 8x8 array

Team 08 of Section 005
William Morgan (wtmorga3) Jake Pope (jmpope3) Joey Woodring (jjwoodri) Pierce Willoughby (pxwillou)
Connect Four

System Testing

Instructions: https://go.ncsu.edu/csc116-ce-stp

Journal

Remember: System tests should only be based on your requirements and should work independent of design.

The following prompts are meant to aid your thought process as you complete the system testing portion of this exercise. Please review your list of requirements and respond to each of the prompts below. Feel free to add additional notes.

1. What does input for the software look like (e.g., what type of data, how many pieces of data)?
    - The command line will prompt the user to give a column to place a chip in
    - One piece of data
2. What does output for the software look like (e.g., what type of data, how many pieces of data)?
    - The program will display the grid with chips in previously selected locations
    - If the selected column is full, give an usage message letting the user know that they cannot place in that column
    - If four chips are placed in a row, output a game-winning screen
    - Pieces of data: the grid plus however many chips are in the grid
3. What equivalence classes can the input be broken into?
    - Input can be broken down into full columns or empty columns
4. What boundary values exist for the input?
    - Valid values are between 1-8(inclusive)
5. Are there other cases that must be tested to test all requirements?
    - When the board is full but no winning patterns are found
6. Other notes:
    - N/A

System Test Cases

Use your notes from above to complete the system test plan section of the formal documentation by writing system test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement. For example, if you have 6 requirements, you should have at least 6 six test cases. **Make sure your test cases are non-redundant, specific, and repeatable!**

Test ID	**Description**	**Expected Results**	**Actual Results**
boundaryValueTest0	Prerequisite: program started Player 1 input column: 0	CLI Prints: “Usage: Column input must be between 1-8” “Player 1 input column:”	CLI Prints: “Usage: Column input must be between 1-8” “Player 1 input column:”
boundaryValueTest1	Prerequisite: program started Player 1 Input column: 1	CLI Prints: Grid with chip placed on column 1 at lowest row possible “Player 1 has placed 1 chips and has 1 in a row” “Player 2 input column:”	CLI Prints: Grid with chip placed on column 1 at lowest row possible “Player 1 has placed 1 chips and has 1 in a row” “Player 2 input column:”
boundaryValueTest8	Prerequisite: program started Player 1 Input column: 8	CLI Prints: Grid with chip placed on column 8 at lowest row possible “Player 1 has placed 1 chips and has 1 in a row” “Player 2 input column:”	CLI Prints: Grid with chip placed on column 8 at lowest row possible “Player 1 has placed 1 chips and has 1 in a row” “Player 2 input column:”
boundaryValueTest9	Prerequisite: program started Player 1 Input column: 9	CLI Prints: “Usage: Column input must be between 1-8” “Player 1 input column:”	CLI Prints: “Usage: Column input must be between 1-8” “Player 1 input column:”
testGameWon	Prerequisite: program started Player 1 Input column: 1 Player 2 Input column: 7 Player 1 Input column: 1 Player 2 Input column: 6 Player 1 Input column: 1 Player 2 Input column: 7	CLI Prints: “Player 1 has placed 4 chips, and has 4 in a row” “Player One wins!” “Player One has 2 wins!” “Press N for a new game or Q to quit: “	CLI Prints: “Player 1 has placed 4 chips, and has 4 in a row” “Player One wins!” “Player One has 2 wins!” “Press N for a new game or Q to quit: “

	Player 1 Input column: 1		
testGameOverNoWin	Prerequisite: program started Players give valid inputs 32 times each Board is filled with turns that do not end with the winning pattern	CLI Prints: The grid with no winning patterns “Game over! No player wins” “Press N for a new game or Q to quit”	CLI Prints: The grid with no winning patterns “Game over! No player wins” “Press N for a new game or Q to quit”
testColumnFull	Prerequisite: program started Player 1 Input column: 1 Player 2 Input column: 1 Player 1 Input column: 1 Player 2 Input column: 1 Player 1 Input column: 1 Player 2 Input column: 1 Player 1 Input column: 1 Player 2 Input column: 1 Player 1 Input column: 1	CLI Prints: “Column 1 is full.” “Player 1 Input column:”	CLI Prints: “Column 1 is full.” “Player 1 Input column:”
testColumnEmpty	Prerequisite: program started Player 1 input column: 3	CLI Prints: New Grid with chip placed in column 3 lowest row possible “Player 2 input column:”	CLI Prints: New Grid with chip placed in column 3 lowest row possible “Player 2 input column:”
testInvalidPostGameOption	Prerequisite: program started Game ends “Press N for a new game or Q to quit: G”	CLI Prints: “Invalid input” “Press N for a new game or Q to quit:”	CLI Prints: “Invalid input” “Press N for a new game or Q to quit:”

Team 08 of Section 005
William Morgan (wtmorga3) Jake Pope (jmpope3) Joey Woodring (jjwoodri) Pierce Willoughby (pxwillou)
Connect Four

Design

Instructions: http://go.ncsu.edu/csc116-ce-design

Journal

Remember: You still will not be writing code at this point in the process.

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

1. List the nouns from your requirements/analysis documentation.
    - Chips
    - Game board
    - Connect Four

2. Which nouns potentially may represent a class in your design?
    - Chips
    - Game board

3. Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
    - X and O for the type of chip to be played (player 1 and player 2)
        - Part of chip class
    - The array of chips that makes up the game board
        - Game board class
    - Size of game board, number of columns and rows
        - Game board class
    - Array for each column of chips
        - Game board class

4. Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types, in addition to a class that represents client code.
    - Design with classes for chip and game board
        - Pros:
            - Easily distinguish between chip and board using chip class to check for winning patterns on the board
        - Cons:
            - Potentially harder to make use of just the column
    - Design with classes for chip, board and, column
        - Pros:

- Further help distinguish between each column to check whether it is a valid input for the user
- Cons:
  - Potentially too redundant for a whole class for the column

5. Which design do you plan to use? Explain why you have chosen this design.
   - We plan to use the design with the chip class and the board class
     - We chose this design because it is a simpler representation of the interacting objects within a standard game of connect four, and we can adequately complete all of the requirements using this design

6. List the verbs from your requirements/analysis documentation.
   - Checking if the pattern on the board has winning patterns
   - Getting the type for a certain chip
   - Checking if a certain column is full
   - Dropping a chip down a selected column

7. Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
   - isWinningGame
     - Board class
   - getType
     - Chip class
   - isColumnFull
     - Board class
   - dropChip
     - Board class

8. Other notes:
   - N/A

Prior to starting the Design formal documentation, you should show your Design Journal answers to the teaching staff.

Team 08 of Section 005
William Morgan (wtmorga3) Jake Pope (jmpope3) Joey Woodring (jjwoodri) Pierce Willoughby (pxwillou)
Connect Four

Software Design

<<Use your notes from above to complete this section of the formal documentation by planning the classes, methods, and fields that will be used in the software. Your design should include UML class diagrams. Make sure your POJOs (Plain Old Java Objects) and classes preserve encapsulation.>>

Chip.java
Fields: -type: char
Methods: +getType(): char

Board.java
Fields: -board: chip[][] -size: int -column: chip[]
Methods: +isColumnFull(Column): boolean +dropChip(Chip, int): void +isWinningGame(): boolean +toString(): String

Team 08 of Section 005
William Morgan (wtmorga3) Jake Pope (jmpope3) Joey Woodring (jjwoodri) Pierce Willoughby (pxwillou)
Connect Four

Implementation

Instructions: http://go.ncsu.edu/csc116-ce-imp

Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation.
    - We will create an object called Chip
        - Create private field for type of chip
        - Create constructor for this object
        - Create getter method for this type
    - We will create an object for the board
    - Use loops to iterate through the 2d array of the board to check for a game winning pattern
    - 555

- Other notes:
    - <<Insert notes>>

Implementation Details

<<Use your notes from above to write code and complete this section of the formal documentation with a README for the user that explains how he/she will interact with the system (e.g., how to compile and run).>>

README:

1. How to compile and run this program
    a. Make sure you are in CE directory
    b. Compile: javac -d bin -cp bin src/ConnectFour.java
    c. Run: java -cp bin ConnectFour
2. CLI will prompt the user for how many chips in a row wins the game
3. The CLI will display the grid based on how many chips the user selects
4. The CLI will ask player 1 to drop their chip on a desired column
5. CLI will then prompt player 2 to drop their chip on their desired column
6. If the column is full, CLI prints “Column [x] is full” and reprompt the player for a different column
7. If the board fills up with no wins, CLI prints “Game over! No player wins” and displays options to start a new game or to quit
8. If a winning pattern is found, CLI prints “Player [x] wins! Player [x] has [x] wins” and displays post game options

Team 08 of Section 005
William Morgan (wtmorga3) Jake Pope (jmpope3) Joey Woodring (jjwoodri) Pierce Willoughby (pxwillou)
Connect Four

Testing

Instructions: http://go.ncsu.edu/csc116-ce-test

Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

1. Have you changed any requirements since you completed the system test plan? If so, list changes below and update your system test plan appropriately.
    - Nothing changed from the original requirements that would need us to update the system test plan. Additional helper methods were required for the board class for use in the unit and integration testing but those changes will not be reflected in the system test plan. The one change that was made in order to fulfill the requirements was adding the functionality of keeping track of the number of chips placed and the number each player has in a row until they have won a game.

2. List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
    - Chip
        - Boundary values: none
        - Only valid types are X and O
        - We should test the equals method with different objects to ensure that it checks for every instance
        - We will also test the getter methods
    - Board
        - Boundary values: on the edges of the array, for example, values 1-8 for the columns
        - Equivalence classes: values below the accepted value, accepted values, values above accepted range for column input
        - We will test the isWinningGame method thoroughly and check for a vertical, horizontal and both diagonal win conditions.
        - We will also test the toString method to ensure that the board is being printed to the command line properly.
        - We will test the getInARow method to ensure that the highest value is chosen out of the array of values
        - We will test the dropChip method to ensure that the chips are being dropped in the lowest row possible.
        - We will finally test the isColumnFull method for each column and a full board to ensure that chips cannot be placed in a full column

3. Other notes:
    - <<Insert notes>>

Team 08 of Section 005
William Morgan (wtmorga3) Jake Pope (jmpope3) Joey Woodring (jjwoodri) Pierce Willoughby (pxwillou)
Connect Four

Testing Details

<<Use your notes from above to write your test programs and complete this section of the formal documentation by creating a list of your test programs along with descriptions of what they are testing. You will also complete the system test plan by running the program and filling in the Actual Results column.>>

Chip Test

- This program is testing the chip object that each player will be assigned to
- The getter method is tested with the two acceptable inputs for the type of player
- The equals method is tested with an object against the same instance of itself (returns true)
    - Equals method is also tested against a different object with the same type (returns true)
    - Equals method is tested against a different object with a different type (returns false)
    - Equals method is tested against the two different player chip objects (returns false)

Board Test

- This program is testing the board object that will house the logic of the game and the methods needed to run it
- The dropChip method is tested for 3 different instances of player objects dropped into the array
    - The first instance is of player 1 in column 1
    - The second instance is of player 2 in column 1
    - The third instance is of player 1 in column 2
- The isColumnFull method is tested on column 1 of the board, which was fulled with player objects (returns true)
    - The method was also tested with an empty column (returns false)
- The isWinningGame method was also tested with each equivalence class
    - An empty board was tested (returns false)
    - A vertical winning game was tested in column 1 (returns true)
    - A horizontal winning game was tested in row 1 (returns true)
    - An upward diagonal winning game was tested (returns true)
    - A downward diagonal winning game was tested (returns true)
- The toString method was tested with a random board
- The getInARow method was tested using the same boards as the isWinningGame method and each board should have a return value of 4

Presentation

Instructions: http://go.ncsu.edu/csc116-ce-presentation

Preparation

The following prompts are meant to aid your thought process as you complete the presentation portion of this exercise. It is recommended that you examine the previous sections of the journal and your reflections as you work on the presentation as it is likely that you have already answered some of the following prompts elsewhere. Please respond to each of the prompts below and feel free to add additional notes.

1. Give a brief description of your final project
    - Our project is a game of Connect 4. The user is asked to input a size for the board. The board is made up of a square array with a length of size inputted by the user. A winner is declared when a number, half of the array length, of chips of the same type are consecutive. Player 1 is then prompted to enter a column of the board to drop a chip into. After valid input, Player 2 is then prompted to enter a column of the board to drop a chip into. After each input, the board is displayed on screen. This is repeated until one player has the required number of chips consecutively to win. Once a winner is declared by the program, the winner, player 1 or player 2, is displayed.

2. Describe your requirement assumptions/additions.
    - A grid needs to be displayed with the ability to show chips placed. When one chip type has been placed four times in a row it wins. This was later changed to whatever game board size the user picks, which determines the winning length. A win can be from a vertical, horizontal, or diagonal line. When a chip is being placed, it can't be placed where another chip has already been placed.

3. Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
    - One option we thought about was creating a GUI. A GUI would be a nice display and make for easier playability. However, we ultimately decided against having a GUI. This is because our knowledge of GUIs is limited and a GUI is not a requirement for this project.
    - We chose to split this project into three classes. Those being the Chip, Board, and ConnectFour. Chip creates a chip object of type X or O depending on the player using it. Board creates a board object which is an array that can be filled with chips. ConnectFour runs the game, keeps score, and declares a winner.

4. How did the extension affect your design?
    - The extension informed us that player scores need to be kept track of. So we implemented a score that displays how many wins each player has.
    - We also learned that we had to allow the user to set the board size and in turn the number of chips required to win.

5. Describe your tests (e.g., what you tested, equivalence classes).

- We tested boundary values, by checking column values right outside the range, and checking that the program reads the incorrect input and displays a message to correct the user. The column values on the very end of the range were also tested so that the correct range was being used. Tests were done to make sure that the program could read when a column was full, so it wouldn't place a chip. A test was done for empty columns. For a winning game, a test was done to check that the correct menu prompt displays asking the user if they want to play a new game or quit. We did a similar test for a no-winner game, which displays a message telling the user there is no winner and asking the user if they want a new game or to quit. Invalid user input was also tested, in case the user puts in the wrong input when asked for a new game or quit.

6. Include a detailed, insightful discussion of at least 2 lessons learned about the software process based on the comprehensive exercise.
   - We learned the importance of keeping track of project requirements when implementing code to fulfill all the requirements. This is important because it allows us to create programs that satisfy the requirements before large changes must be made to meet all requirements.
   - We learned the importance of integration and system testing this is what proves that the code meets a portion of the requirements of this exercise. This is important because it makes sure that the program that you have created is fully functional and operates as expected with the project requirements that are given.

7. What functionalities are you going to demo?
   - We will show a game of connect four in which the board will be an 8x8 grid. We will show this game until completion, meaning we shall either have a winner or a full board.

8. Who is going to speak about each portion of your presentation? (Recall: Each group will have seven minutes to present their work; the minimum length of group presentation is five minutes. Each student must present for at least one minute of the presentation.)
   - Joey Woodring: Introduction and first two slides
   - Jake Pope: Requirements and Design slides
   - Pierce Willoughby: Testing and lessons learned
   - William Morgan: Demo

9. Other notes:
   - <<Insert notes>>

<<Use your notes from above to complete create your slides and plan your presentation and demo.>>

Grading Rubric and Final Submission

http://go.ncsu.edu/csc116-ce-grading